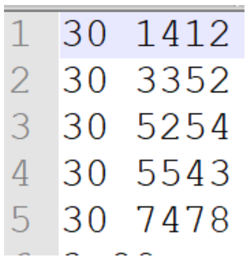
Homework #5: Link Analysis & Social Networks

Due: April 22, Sunday
100 points

In this homework, we consider implementing two algorithms each on topics – Link Analysis and Social Networks.

Task 1 [30 points, Link Analysis - HITS]:

In this task, you are asked to implement HITS algorithm in **spark**: **FirstName_LastName_hits.py** using parallel implementation logic from pagerank.py to calculate the hub and authority on the given input graph in Wikipedia vote network data - **Wiki-Vote.txt**. Data is taken from http://snap.stanford.edu/data/wiki-Vote.html. The network contains all the Wikipedia voting data from the inception of Wikipedia till January 2008. Nodes in the network represent Wikipedia users and a **directed edge** from node i to node j represents that user i voted on user j.



```
30 1412
30 3352
30 5254
30 5543
30 7478
```

30 1412 means there is a directed edge from node 30 to node 1412.

The pagerank code – **pagerank.py** can be found under the <your spark installation directory>/examples/src/main/python. A copy is also provided with this handout. The code uses teleportation, which we don't need for our task. This is what you need to do:

1. Generate the adjacency list of graph
2. Start with a hub vector of all 1's – $\boldsymbol{h}^0$
3.
    a. Compute Authority vector - $\boldsymbol{a}^0 = \boldsymbol{L}^T\boldsymbol{h}^0$. Recall that this is similar to PageRank computation and the Spark code computes this by distributing the pageranks of a node using forward links of the node. See slide #59. Remember to normalize the authority vector so that the largest value is 1.
    b. Using the computed Authority vector, compute new hub vector $\mathbf{h}^1 = \boldsymbol{L}\boldsymbol{a}^0$. Think about how you can implement this based on the hint given in class. Again, normalize the hub vector so that the largest value is 1.
4. Repeat step 3 for given number of iterations using the new values of hub and authority vectors computed in each iteration

5. Output the authority and hub values you got from last iteration to relevant files.

Execution format:

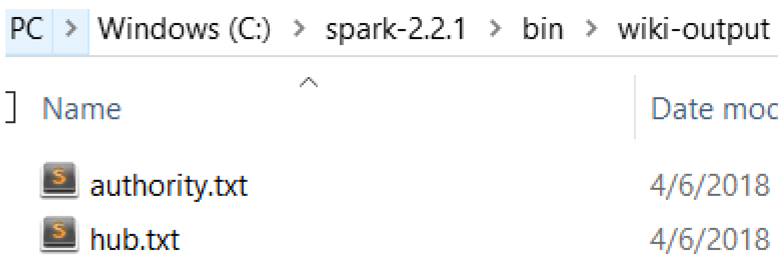
spark-submit FirstName_Lastname_hits.py <input_file> <iterations> <output-dir>

- <input_file> is either Wiki-Vote.txt or another graph file (Of similar or small size) that has directed edges
- <iterations> is the number of iterations your hits algorithm will run for. Maximum 5 iterations.
- <output-dir> is the directory name where you will create your two files in:
    - **authority.txt** will have node and its authority with comma separated values
    - **hub.txt** will have node and its hub with comma separated values

Example execution:

spark-submit hits.py Wiki-Vote.txt 5 wiki-output

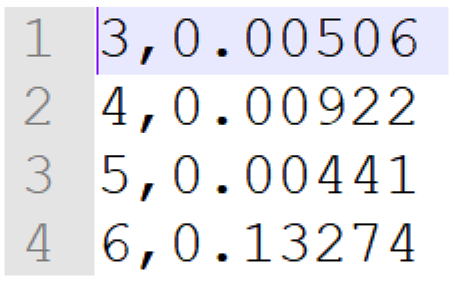
wiki-output directory needs to be created (If doesn't exist) and should have two files – authority.txt and hub.txt like below.



```
authority.txt    4/6/2018
hub.txt          4/6/2018
```

Output Format:

Each line in **authority.txt** needs to have **nodeId,authority** with nodeId in ascending sorted order, and authority in 5 floating points - **"{0:.5f}"**.format. Similarly, **hub.txt** needs to have **nodeId,hub** with nodeId in ascending sorted order, and hub in 5 floating points. Example (Not an actual) output for both:



```
3,0.00506
4,0.00922
5,0.00441
6,0.13274
```

Task 2 [70 points, Social Networks - Spectral Clustering]:

In this task, you are asked to implement spectral clustering algorithm, where a graph is continuously partitioned to get a desired number of clusters in Python – **FirstName_LastName_spectral.py** using graph library – **networkx.** Read about networkx here - https://networkx.github.io/documentation/stable/index.html# about installation, how to use etc. With networkx, you can

1. Construct a graph from edges
2. Compute degree of each node that you would require for Degree Matrix
3. Modify the graph etc.

You would also need library function to calculate Eigenvalue / Eigenvector of Laplacian Matrix, for which you can use **numpy**.

Spectral Algorithm works in 3 steps: For user specified number of clusters – k (Minimum 2)

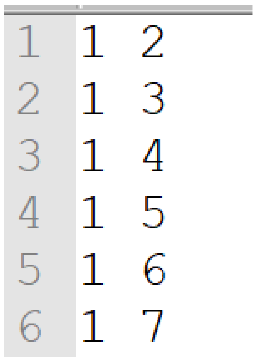
1. Construct Laplacian Matrix L for the given graph
2. Find Eigenvalues and Eigenvectors for the Matrix L using numpy. Pick the second smallest Eigenvalue and corresponding Eigenvector. Split the Eigenvector at 0 (Each element mapped to each node) into 2 clusters – i.e. place positive elements and negative elements (mapped vertices) into two clusters.
3. Based on k, repeat steps 1 and 2. The way you pick next cluster to partition is based on which among already identified clusters have more nodes. You disconnect edges that connect nodes from one cluster to another, and build Laplacian Matrix again.

For example:

Let's say there are 7 nodes [1,2,3,4,5,6,7] in a graph, and you are asked to find 3 clusters. You construct a Laplacian Matrix L, compute Eigenvalue and Eigenvector. For second smallest Eigenvalue, you get an Eigenvector that has 3 positive and 4 negative values. So, you get 2 clusters – say [1,2,3] and [4,5,6,7]. You will pick cluster [4,5,6,7] to partition again because it has more nodes than other cluster. You construct a graph for the cluster [4,5,6,7] by removing all edges that connect to cluster [1,2,3]. You repeat the steps of constructing Laplacian Matrix for the new graph, finding Eigenvalue and Eigenvector, splitting the Eigenvector at 0 to get 2 more partitions. You will end up with 3 clusters now – for example [1,2,3], [4,5], [6,7].

Note: You can be ensured that that clusters will always have unequal number of nodes at each step. So, your choice to pick cluster with more nodes than other is always possible.

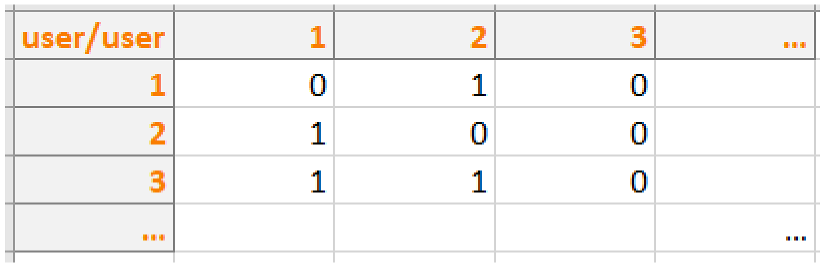
You are given with Facebook (NIPS) data **out.ego-facebook** that has user – user friendships. Data is taken from http://konect.uni-koblenz.de/networks/ego-facebook . A node represents a user. An edge indicates that the user represented by the left node is a friend of the user represented by the right node, and vice versa. You need to consider this as an undirected graph so that user A and user B are friends of one another.



```
1 2
1 3
1 4
1 5
1 6
1 7
```

1 2 is an undirected edge from user 1 to user 2.

Your Adjacency Matrix – A (For Spectral) needs to have sorted userId's on both rows and columns. Same goes for your Degree Matrix.



user/user	1	2	3	...
1	0	1	0	
2	1	0	0	
3	1	1	0	
...				...

Execution Format:

python FirstName_LastName_spectral.py <input_file> <k> <output_file>

1. <input_file> is graph file containing undirected edges
2. <k> is desired number of clusters to be identified. Minimum 2, maximum – A valid k that does not break the code when you have to pick cluster with more nodes than other for partitioning
3. <output_file> is a text file where you will write your output to.

Example execution:

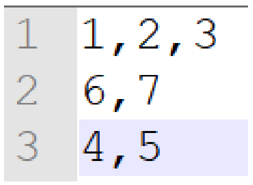
python spectral.py out.ego-facebook 3 clusters.txt

clusters.txt should be created that will have 3 clusters in the output format below

Output Format:

Each line should contain comma separated nodes (user ID's) from each cluster in ascending sorted order. Order of clusters does not matter.

For example:



```
1,2,3
6,7
4,5
```

Submission on Blackboard:

A zip file **FirstName_LastName_hw5.zip** containing Python files FirstName_LastName_hits.py and FirstName_LastName_spectral.py that can be executed with parameters mentioned above. If you are using Python 3, please rename the files to FirstName_LastName_hits3.py and FirstName_LastName_spectral3.py.

Grading Criteria:

1. If task1 result is not sorted on nodeId, there will be 20% penalty. 20% penalty if output format is not followed.
2. If nodes of each cluster in task2 result are not sorted, there will be 20% penalty. 20% penalty if output format is not followed.
3. Program execution time needs to be under 5 minutes for both tasks. 30% penalty otherwise.
4. 10% penalty if naming convention for python scripts is not followed.
5. Assignment is due at 11:59 pm on 04/22. Late submission will have 10% of penalty for every 24 hours that it is late. No credit will be given after 72 hours of its due time.

References:

1. http://snap.stanford.edu/data/index.html
2. J. Leskovec, D. Huttenlocher, J. Kleinberg. Signed Networks in Social Media. CHI 2010.
   J. Leskovec, D. Huttenlocher, J. Kleinberg. Predicting Positive and Negative Links in Online Social Networks. WWW 2010
3. Facebook (nips) network dataset -- KONECT, April 2017. [http]
4. Julian McAuley and Jure Leskovec. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems*, pages 548--556. 2012.
5. http://konect.uni-koblenz.de/

We will be using Moss for plagiarism detection. Do not share your code with anyone, or copy from others!